# Method and Type Descriptors

Here’s how a method (.ctor) is declared:

.method public specialname rtspecialname instance void .ctor() cil managed

Now here’s how methods are declared on that class:

.method assembly static class EmitDebugging.Tests.DynamicBuilders GetDynamicBuilders(string codeFile) cil managed

.method public instance void CreateMethodWithFaultBlock() cil managed

Notice that:

* The static/instance modifier is always given.
* The return type is given. If it’s void or a “special” type (i.e. int32, string, etc.), only the “special” type descriptor is given. Otherwise, the full name of the type is given. Also, the “class” or “valuetype” is given before the type descriptor to indicate if it’s a reference or value type.
* The argument names are given.
* The “cil managed” is at the end of the declaration.

Now, calling methods goes like this. Here’s how a (static) method on the class is called:

call class EmitDebugging.Tests.DynamicBuilders EmitDebugging.Tests.EmitUtilities::GetDynamicBuilders(string, bool, bool, valuetype [mscorlib]System.Reflection.TypeAttributes)

Here’s how calling a (static) method within the assembly is called:

call class EmitDebugging.Tests.DynamicBuilders EmitDebugging.Tests.EmitUtilities::GetDynamicBuilders(string)

And here’s how an instance method is called:

callvirt instance !0 class [mscorlib]System.Collections.Generic.IList`1<class [EmitDebugging]EmitDebugging.TypeDebugging>::get\_Item(int32)

In the first two cases, the static modifier is not given, but for the 3rd one, “instance” is given. The arity is also expressed before the return type. In all three cases, the type is namespace + name, and if the type is outside of the assembly, the assembly name goes before the type encased in [assemblyname]. Argument names are not given; just the type.

So here is the general rule for piecing together a method declaration:

[MethodDeclaration] [Attributes] [Modifier] [Arity] [ReturnType] [TypeDescriptor]::[MethodName]([Arguments]) [ConventionAttributes]

Where:

MethodDeclaration: .method. Only needed for a method declaration

Attributes: public, specialname, etc. Only needed for a method declaration

Modifier: static/instance. Only static is used for a method declaration. Instance is always used.

Arity: !0, etc. Unknown if needed in both declarations and invocations.

ReturnType: [TypeDescriptor]

TypeDescriptor: [assembly]typenamespace+typename. Assembly is only needed if the type is outside of the current assembly. Some types are “special”: string, int32, etc.

MethodName: self-describing. Note that the TypeDescriptor:: prefix is only needed for invocations, not declarations.

Arguments: list of arguments. Declarations use a name (if available). In both cases, [TypeDescriptor] is used for each argument.

So…create MethodDescriptor and TypeDescriptor classes.

## Generic Arguments

If I invoke a generic method:

IsGenericMethod = true

IsGenericMethodDefinition = False

If I “declare” a generic method:

IsGenericMethod = true

IsGenericMethodDefinition = true

By invoking I declare the generic values. Then:

?method.GetGenericArguments()

{Length=1}

(0): {Name = "Guid" FullName = "System.Guid"}

With a declaration:

?method.GetGenericArguments()

{Length=1}

(0): {Name = "T" FullName = Nothing}

So